Final Report - More for Less

Team Members: (in alphabetical order of last name)
Yang Li (yl4324) Quanxing Liu (ql2368)
Ruibin Ma (rm3708) Junlin Song (js5564)

Abstract

We built a promotion web app, More for Less, using AWS technologies. With More for Less, buyers can buy things more economical while sellers can realize more potential sales. Sellers can publish deals, while buyers can search the deals or get recommendations.

Links

Youtube Video: https://youtu.be/cbCQqbVI-YY

Project proposal:
https://docs.google.com/document/d/1XVHFsBS0pa7Pmq7rk2QIM7qUboJEd6gSCi-fy2bjt6k/edit?usp=sharing

Presentation slides:
https://docs.google.com/presentation/d/10OSXPRKb5dKd_vGeRYBCPTaxIhbMOvQ6dz4IgdZzRMo/edit?usp=sharing

Github repo: https://github.com/liuqx0717/COMS6998

Project website: https://moreforless.liuqx.net

Introduction

Customers and merchants - one to sell, one to buy - seem at odds with each other originating from the nature of business activity. For the same product, customers always wants it cheaper, while merchants trying to get the most out of it. With More for Less, buyers can buy things more economical while sellers can realize more potential sales.

As a promotion app, More for Less focus on maximizing potential sales for shop owners while benefiting customers with exclusive discounts. Take food for example. According to a report by the United States Department of Agriculture, which tracks food loss, 10 percent or 43 billion pounds of the 430 billion pounds of the edible and available food supply went uneaten at the retail levels in the United States in 2010. Just imagine when some food in your nice and fancy bakery are going to finish their shelf life, would you like to throw them away without doing anyone good, or more likely, thinking about if you and somebody can do each other a favour - you offer a discount and he buys it - to save the soft croissant from rotting in trash bin. Sounds nice? That’s what More for Less is for, and it’s only one chapter of the whole story.

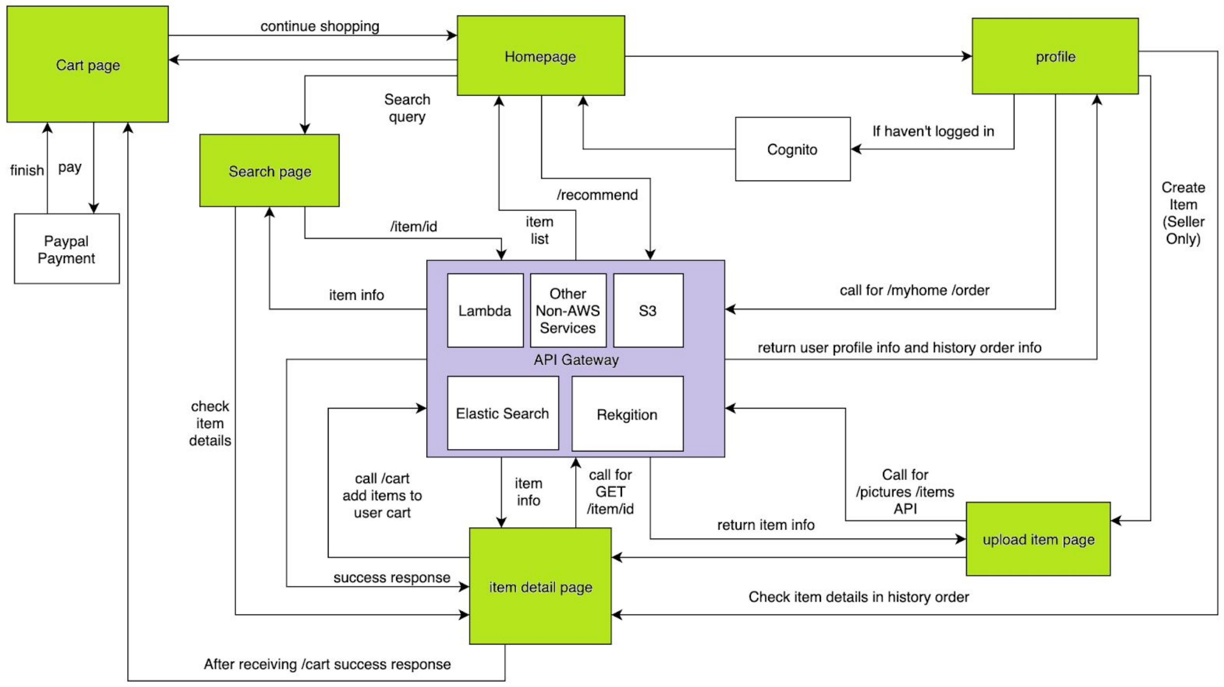
With More for Less, you may explore discounts from a wide variety of categories. From bakery to bars, from pharmacy to wholesale, as long as you can think of it, More for Less may have it. Offering an app where all promotions can be displayed by category or location, More for Less is aimed at providing the most efficient way for you to find the most valuable things you are looking for at the moment, with less cost when you preferred.

In More for Less, Users can register their accounts. Registered sellers can post discounted items on the platform, and manage them later on. They may add name, location, category, description, price, and some pictures to their posts. Buyers can search the items by location, category, etc. to find the discounted items they are interested in. They may also search by voice or images. Customers will get recommendations based on their past activities and current locations. Registered buyers can order items(coupons or goods) and pay for them online, and receive the receipt confirmation via Email / Text message.

More for Less is built on AWS (Amazon Web Service). We used microservice architecture, which decouples the components, increases scalability, expedites debugging and deploying. The frontend is a static website built with bootstrap, hosted on Amazon S3. The API, which enables the communication between frontend and backend, is specified using Swagger. The backend involved API Gateway, Lambda, DynamoDB, Elastic Search, Rekognition, Cognito and Route 53.

Architecture

The diagram below gives a general idea of the architecture of our application.



Check original-size architecture diagram here:

https://raw.githubusercontent.com/liuqx0717/COMS6998/master/proj/Workflow%20-%20color.png

Generally speaking, our design follows the request-response model, as shown in the diagram.

The five green rectangles indicates our five frontend pages (user interface), which are hosted at AWS S3 behind Amazon CloudFront using custom domain name (https://moreforless.liuqx.net/). These frontend pages are connected to multiple backend services through API Gateway (the purple rectangle in the center in the above diagram), while the microservices are powered by Elastic Search, lambda functions, etc. These services are called microservices because they run independently of each other, and they are integrated into one API Gateway endpoint as different resources and methods.

Arrows and corresponding text descriptions shows how frontend interacts with APIs and the related user actions. Basically, the user interface sends requests and get responses from APIs.

Besides, we use Amazon Cognito to handle access control on each page.

Implementation

Frontend Summary

We built a static website as the frontend of our app. We used bootstrap as the basic framework for responsive layout. We used cookies to store identification tokens, attach the token to HTTP headers when performing AJAX operations, thus accessing the resources that require authentication. We bind the S3 endpoint to our custom domain, moreforless.liuqx.net.

Item Details Page (single-product.html)

This page shows all information of an item. Section 1 is a slideshow that cycles through the pictures of the items, implemented using Bootstrap Carousel component (library). Section 2 shows thumbnails. Section 3, the “Add to Cart” button, triggers a function onclick. That function calls PUT /cart and sends item_id with user access token, which results in a wishlisted order in DynamoDB. Section 4 shows some text attributes of the item.

The information of the item is got through API path /items/{id}, while the id is got through query string in the url (/single-product.html?item_id=<item_id>). An item object as the response of the API has the following format:

```
Item:
 type: "object"
 Required:
 - "title"
 - "picture"
 - "price"
 - "stock"
 properties:
         id:
                 type: "string"
         Available:
                 type: "integer"
         sellerId:
```

```
        type: "string"
    sellerInfo:
        Type:
        $ref: "#/definitions/UserProfile"
    title:
        type: "string"
    Description:
        type: "string"
    imageUrl:
        type: "array"  # file name (a.jpg)
    Items:
        type: "string"
    category:
        type: "array"  # user defined keywords
        items:
            type: "string"
    Tag:
        type: "array" # rekognition
        items:
            type: "string"
    price:
        type: "number"
    prvPrice:
        type: "number"
    stock:
        type: "integer"
    Location:
        Lon:
            type: "number"
        lat:
            type: "number"
```

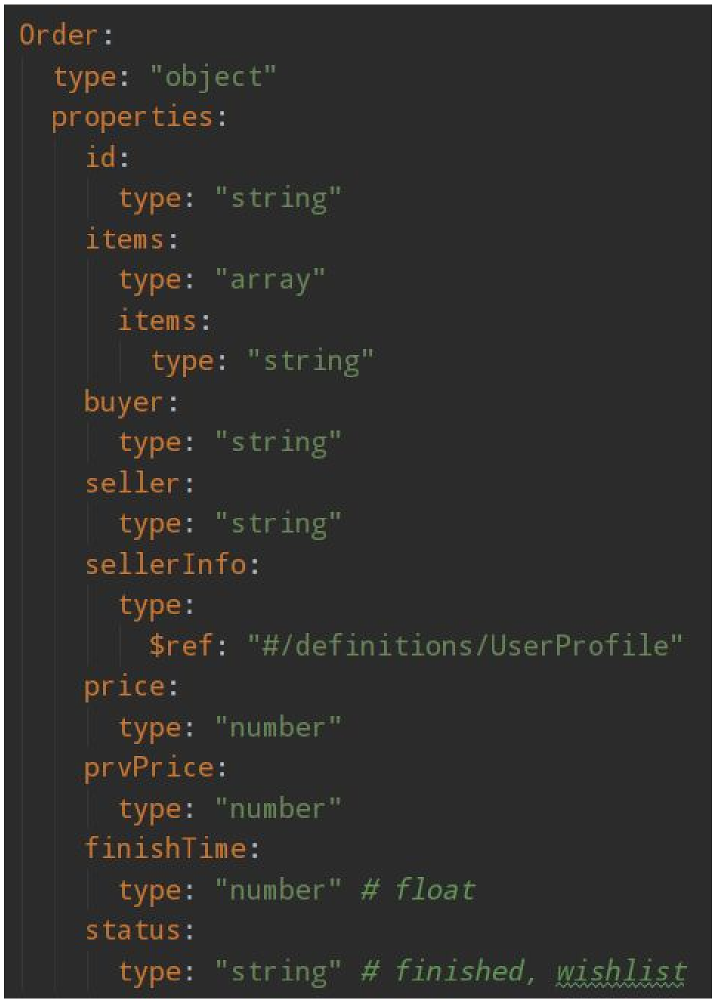
Asynchronous call is used to get item information, to ensure loading speed of static content under low network bandwidth.

Cart Page (cart.html)

This page splits the items in the cart by their sellers, shows item thumbnail, title and price, and calculate subtotals for each seller. If user clicks button “Pay and Finalize”, then he/she will be redirected to the Paypal link that the seller submits as part of profile.

The click also results in a call to POST /cart, which marks a waitlisted order as finished, and an “order” object will be created and will be shown in user’s history orders at profile page. Once the order is marked as finished, it will no longer be shown in user’s cart.

An “order” object has the following format:


```yaml
Order:
  type: "object"
  properties:
    id:
      type: "string"
    items:
      type: "array"
      items:
        type: "string"
    buyer:
      type: "string"
    seller:
      type: "string"
    sellerInfo:
      type:
        $ref: "#/definitions/UserProfile"
    price:
      type: "number"
    prvPrice:
      type: "number"
    finishTime:
      type: "number" # float
    status:
      type: "string" # finished, wishlist
```

Profile page (profile.html)

This page has four main functions.

The first one is that showing user's basic information, including email, phone number and address. This function calls for /myhome API to get the specific user's information. Similar to other parts, this request contains header with access token to identify the user. The backend then retrieve the corresponding user's profile data from database and return it to frontend. The frontend show the profile data by appending content to predefined block in HTML.

The second function is that showing history orders of user. We call for /order API to get user's shopping history. The procedure is the same as the first function. But in this part, in order to show all the history completely, we set AJAX attribute asycn=false. This makes the webpage load a little bit slower but not much.

The third function of this page is that updating users' profile information. User can input their information to update it.

The last one is that checking the current user's type and generate the entry of "upload-item" dynamically. When the user go to this page, the frontend will check the user's type. If the current user is seller, the HTML will generate the button to access the Upload-item page.

Upload item page (Upload-item.html)

Seller access this page to create an item with pictures, introduction and other basic information. We call two APIs: /picture and /item. By accessing /picture API, frontend saves the pictures of the item and gets the image url of the uploaded pictures. In order to get all image urls before uploading the total information, et AJAX attribute asycn=false.

Backend Summary

Backend implementation of More for Less can be divided into data management, functionality and API implementation. Details are illustrated as follows.

Data Management

User Management

To enhance the security and reliability of user pool, AWS Cognito is incorporated into More for Less. Users can sign up with email and phone number and later on verify it, and verified email is primary and is the prerequisite for using More for Less service. Before starting the journey of More for Less, users must choose their type as either buyer or seller, which is crucial for delivering service in the future. Users are given the privilege of modifying their personal information whenever they want, including and not limited to display name, phone number, address and paypal url for sellers for check-out implementation. Users are recognised and located by More for Less with a unique access token generated by AWS Cognito every time they log in to the website. Privacy is well preserved as only access token are passed through different services provided by More for Less and all personal information will be processed in backend without sending to other services.

Item Management

The key part of More for Less is items, i.e. things sellers post to the website for sale. To ease the load of recommendation and searching, we decided to store the information of items, including its ID, name, description, price, etc. in ElasticSearch. One thing to mention is that pictures of items are uploaded to S3 bucket before their links are returned and stored in ElasticSearch along with information of other fields. This implementation greatly improved the performance and eased the burden of functionality implementation, while at the same time offers integrity, availability, durability and the ability to revise information of the data.

Order Management

Orders are sets of items, no matter whether it has been finalized or not - if so, they are history orders; if not, they are still in shopping carts. The implementation of combining history order and shopping carts greatly save the cost of developing the website while at the same time offers the ability to support different functions of the website. Apart from a list of items included in the order, other information of orders are order ID that uniquely identifies an order, buyer ID that shows who placed the order, order total price, a flag of status that tells apart unfinished orders with history orders, and finish time - if the order has been finalized - for reference when an order is completed. All information of orders are stored in AWS DynamoDB, with order ID as primary key and two other global index on buyer ID and seller ID - for displaying all relevant orders on the history orders / shopping cart page when a user browses the website.

Functionality

Create Item

When uploading an items, sellers can upload multiple pictures (see Upload Pictures in the next section), set its name, description, category, previous price, current price, and stocks before POST it to the server. In order to implement recommendation based on geographical location, the precious location of items are recorded with transforming address of seller to coordinates of latitude and longitude with SmartyStreet API. The information of the item is then stored in ElasticSearch in the predicided schema.

Upload Pictures

Users can upload multiple pictures when creating an item. The pictures are stored in S3 bucket and a unique link of it is then returned to the frontend before AWS Rekognition detects tags in the pictures to improve the accuracy and relevance of user searching results.

Get and Revise Item Information

This function is called every time an item is displayed on the website. Also user can update information of an item with PUT method.

Shopping Cart

When a buyer views his/her shopping cart, it should be able to contain all items previously saved to the cart and not yet finalized. Four functions are provided for shopping cart, i.e. GET current shopping cart, PUT an item into the shopping cart, DELETE some items from the shopping cart, and POST a shopping cart order to finalize it. These interactions are based on current user's access token and DynamoDB operations such as get_item, put_item, update_item, delete_item and query. GET updates (in case that prices have been updated after the item was placed in shopping cart) and returns all wishlist order associated with current user ID, PUT and DELETE creates/updates/deletes current wishlist of a buyer associated with a certain seller. POST finalizes an order which changes its status flag to *finished* and create the finish time.

Order History

User can view their full history of finished orders with all information about it to provide a coherent and complete record of the life on More for Less.

My Homepage

User can view their information and update it if they'd like to.

Recommendation

Recommendations are based on user's current address and order history (if logged in). The website will pop a request for location when the user opened the homepage to acquire current location. If not allowed, then user's IP address will be used to determine the approximate location so as to provide a privacy-preserved and still relevant recommendation results. If logged in, recommendation is also based on order history of users to better match his/her personal preference.

Searching

User can search items with keywords such as “chocolate”, to which the website will response with a list of items that might include “chocolate” in its title/description/category/tags.

API Implementation

With the help of AWS API Gateway, HTTP requests can easily be directed to trigger Lambda Function and provide service. One thing to notice is that binary information such as pictures can also be processed with API Gateway after setting the allowed body type and decoding with base64

Results

We successfully implemented everything, connected the frontend well to the backend services, and each part is running as expected. As a result, the website is now functioning well. Users can register and login, get recommendations based on current location, add items to cart, make orders and send payments through Paypal links.

Conclusion

In general, we have successfully completed an online platform for buying and selling preferential products with a complete logical structure and good maintainability. All the website is based on the AWS with a serverless architecture, which achieves high scalability. This project is the successful embodiment of the cloud service concept.

Although there is still room for improvement in the front-end appearance of the project, which we will continue to work on in the future. The whole project has fully realized the expected purpose and functions, and has the value of commercialization.